

Lezione n.22

PEERSIM:

UN SIMULATORE DI RETI P2P

Laura Ricci

24/5/2013

alcuni esempi sono tratti dai lucidi
del Prof. Moreno Marzolla

SIMULAZIONE DI SISTEMI P2P

- Un sistema P2P è, in generale, composto da **milioni di peers**
- La valutazione di un nuovo protocollo in un ambiente reale è in generale di difficile realizzazione
 - necessità di disporre di un alto numero di hosts
 - problemi: presenza di NATs, firewalls che ostacolano la comunicazione
 - definizione di ambienti che consentano la gestione di un programma distribuito su larga scala
 - dinamicità
 - ripetibilità degli esperimenti
- Ambienti attualmente disponibili per la valutazione di sistemi P2P
 - **Planet Lab**
 - **GRID 5000**
- Alternativa: utilizzare **simulatori altamente scalabili**

PEERSIM: INTRODUZIONE

- Simulatore open source, basato su JAVA, sviluppato all'Università di Bologna
- Caratteristiche principali:
 - Scalabilità (soprattutto se usato con modalità cycle-based...)
 - Configurabilità
 - **Plug in Simulation**: simulazione definita mediante
 - un insieme di classi 'core' della simulazione
 - un insieme di componenti definite dall'utente che possono essere 'inserite' nel simulatore
 - Simulazione eseguita su un unico host. Il simulatore single-threaded non sfrutta architetture multi-core
- Documentazione
 - JavaDOC ed un insieme di tutorial on line (presenti sul sito)
 - <http://peersim.sourceforge.net/doc/>
 - Articoli con proposte di nuovi overlay valutati mediante Peersim

PEERSIM: INTRODUZIONE

- **Struttura del simulatore**
 - un nucleo minimo che realizza le funzionalità di base della simulazione
 - l'utente può definire nuove componenti
 - l'implementazione di una componente può essere facilmente rimpiazzata con un'implementazione diversa
 - i riferimenti alle componenti definite dall'utente sono raccolti in un **file F di configurazione**
 - il simulatore carica dinamicamente le componenti definite in F
 - **single threaded simulator**
- **Modalità di simulazione**
 - cycle-driven
 - event-driven

PEERSIM: MODALITA' DI SIMULAZIONE

Caratteristiche della simulazione *cycle-driven*

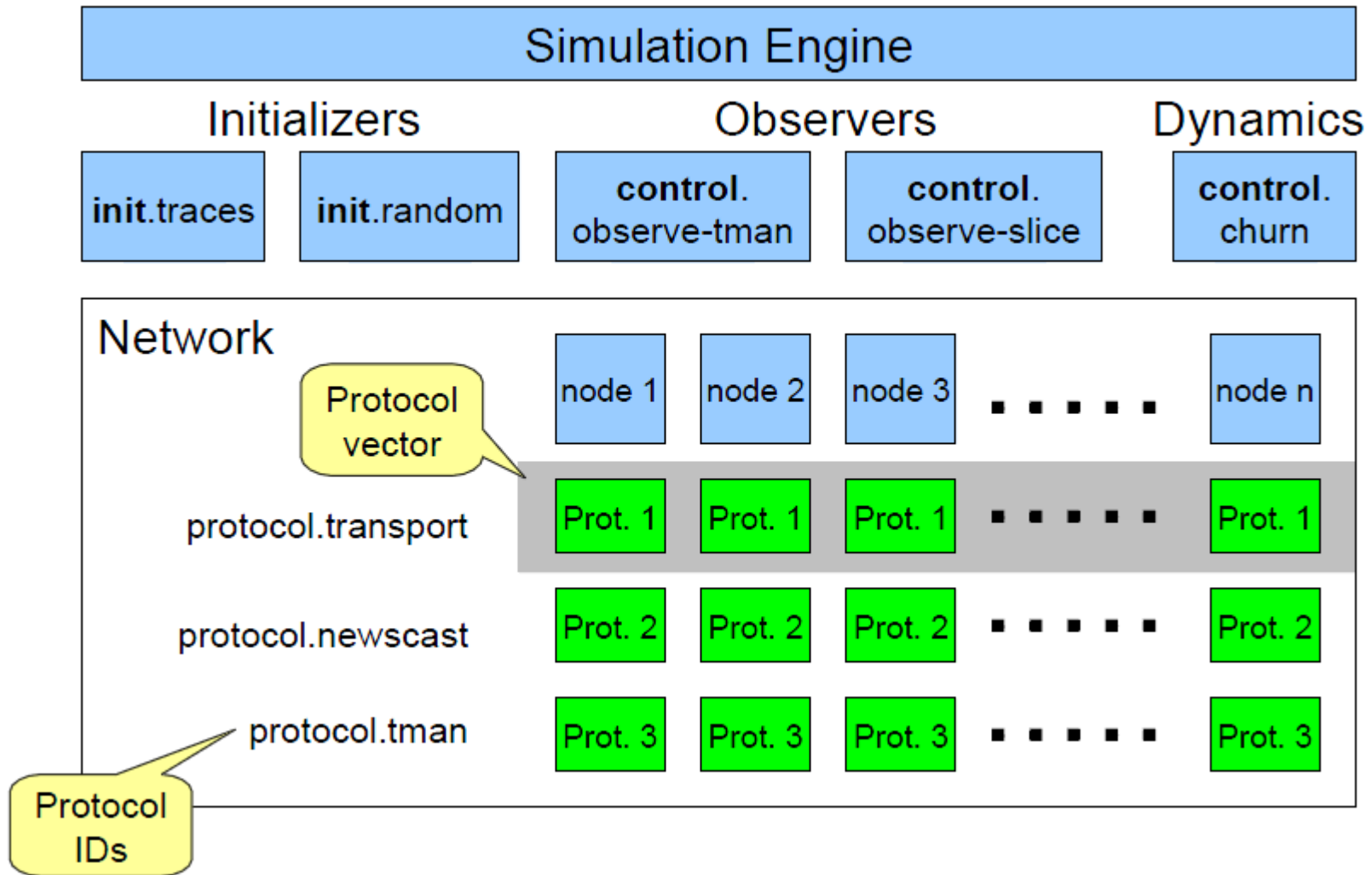
- overlay P2P: insieme di *nodi*
- nodo: '*contenitore*' di *protocolli*
- la simulazione consiste nel considerare in sequenza i nodi e nell'eseguire tutti i protocolli su ogni nodo
- Semplice, un pò 'primitivo', schema sincrono
 - lo scambio di messaggi tra nodi è simulato mediante l'*esecuzione di metodi*
 - non modella il *livello trasporto* dell'overlay (latenze, perdite di messaggi,...)
- utilizzato per analizzare proprietà dell'overlay indipendenti dal livello trasporto
 - crescita del numero di vicini di un nodo all'aumentare del numero di nodi
 - convergenza di *algoritmi di gossiping*
- testato fino a 10^7 nodi

PEERSIM: MODALITA' DI SIMULAZIONE

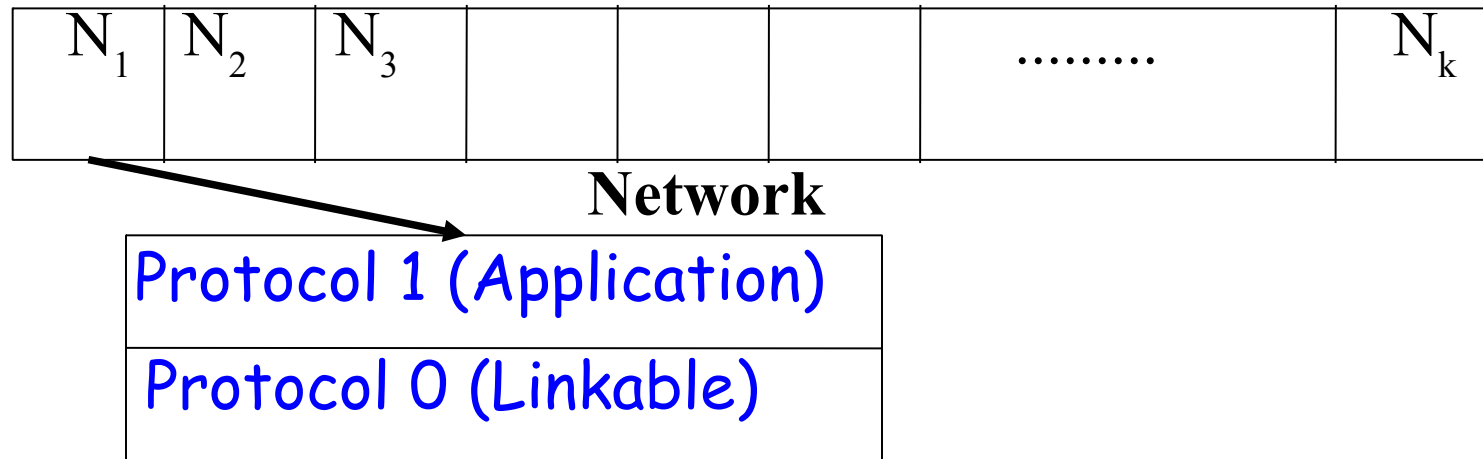
Caratteristiche della simulazione *event-driven*

- più realistica
- modella lo scambio di messaggi tra nodi: send, receive
- eventi= schedulazione dei messaggi ricevuti dai nodi
- l'esecuzione dei protocolli presenti su ogni nodo è guidata dagli eventi
- può essere utilizzata congiuntamente con il simulatore cycle-driven
 - alcuni protocolli schedulati ciclicamente, altri schedulati quando avvengono alcuni eventi
- testato fino a $2.5 \cdot 10^5$ nodi

PEERSIM: L'ARCHITETTURA

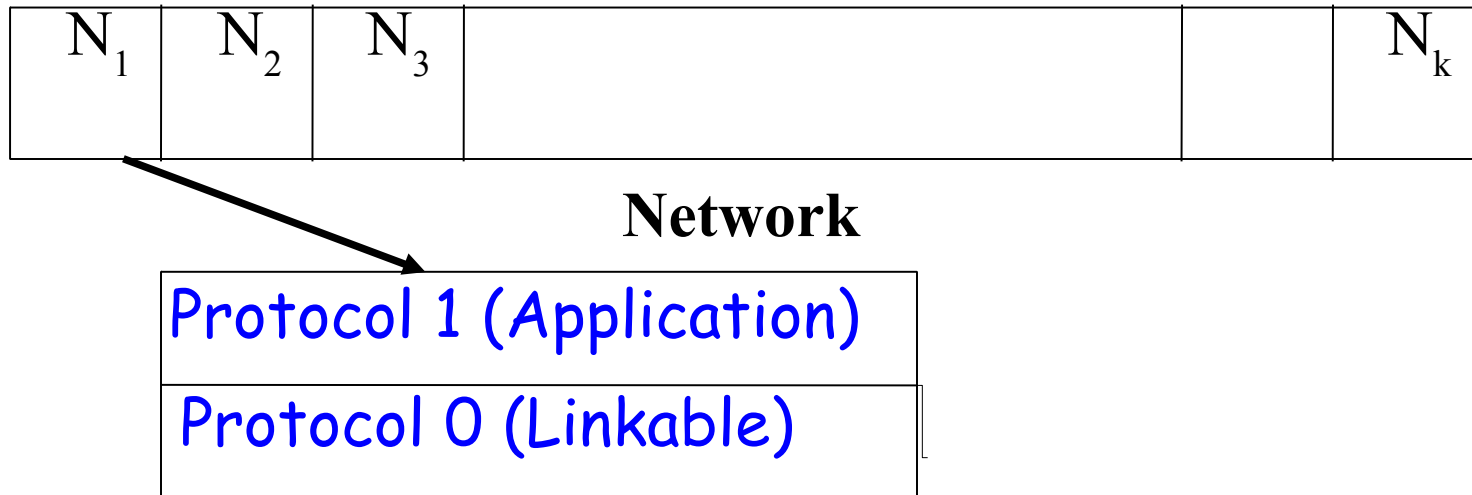


PEERSIM: L'ARCHITETTURA



- **Network** = Vettore contenente riferimenti a tutti i nodi della rete. Viene creata **clonando un nodo prototipo per k volte** (k dimensione della rete)
- **Nodo**
 - identificato da un **PID**
 - contiene uno **stack di k protocolli**, inizialmente assumeremo **k=2**
 - le azioni di ogni nodo sono descritte dai protocolli definiti dal nodo

PEERSIM: NUCLEO MINIMO



- **Protocolli:** specificano il comportamento del nodo, identificati univocamente dal PID
- tutti i nodi eseguono gli stessi protocolli.
- Un caso minimo:
 - un protocollo **Application** definito dall'utente (es: simula il routing)
 - il protocollo **Linkable** = **Contenitore di Nodi**, non eseguibile, rappresenta la **visione della rete posseduta da un nodo** (contatti con i nodi vicini)

LA CLASSE NETWORK

```
public class Network {  
    /** Numero di nodi appartenenti alla rete  
    public static int size ( ) { return len; }  
    /** Restituisce un nodo dato indicizzato da index  
    public static Node get (int index) {....}  
    /** Aggiunge un nodo dalla rete  
    public static void add (Node n) {....}  
    /** Rimuove un nodo dalla rete  
    public static void remove ( ) {....}  
    .....
```

- Network: array globale che contiene tutti i nodi
- PeerSim implementa la classe Network (vedi documentazione)

L'INTERFACCIA NODE

- Nodo: stato+insieme di protocolli eseguiti
- Lo stato di un nodo e le sue azioni vengono descritte mediante uno stack di protocolli
- Ogni protocollo è caratterizzato da un identificatore unico ed è accessibile mediante questo identificatore
- Ogni nodo possiede una vista locale dell'overlay
 - si utilizza una classe che implementi l'interfaccia `Linkable` per memorizzare la vista locale del nodo

L'INTERFACCIA NODE

```
public interface Node extends Fallible, Cloneable
{
    /**
     * Returns the <code>i</code>-th protocol in this node. If <code>i</code>
     * is not a valid protocol id (negative or larger than or equal to the number
     * of protocols), then it throws IndexOutOfBoundsException.
     */
    public Protocol getProtocol(int i);

    /**
     * Returns the number of protocols included in this node.
     */
    public int protocolSize();

    /**
     * Returns the unique ID of the node. It is guaranteed that the ID is unique
     * during the entire simulation, that is, there will be no different Node
     * objects with the same ID in the system during one invocation of the JVM.
     * Preferably nodes
     * should implement <code>hashCode()</code> based on this ID.
     */
    public long getID();

    /* ... */
}
```

L'INTERFACCIA PROTOCOL

```
public interface Protocol extends Cloneable
{

    /**
     * Returns a clone of the protocol. It is important to pay attention to
     * implement this carefully because in peersim all nodes are generated by
     * cloning except a prototype node. That is, the constructor of protocols is
     * used only to construct the prototype. Initialization can be done
     * via {@link Control}s.
     */
    public Object clone();

}
```

- L'interfaccia è molto generale: definite interfacce che la estendono
- Il metodo clone() deve essere riscritto dall'utente in modo da allocare, per ogni nodo, copie diverse delle strutture dati accedute dal protocollo
- Una copia diversa per ogni istanza del protocollo in esecuzione su un nodo diverso

CYCLE DRIVEN PROTOCOLS: INTRODUZIONE

- L'interfaccia `CDProtocol` è utilizzata per definire cycle-driven protocols, protocolli le cui azioni devono essere eseguite da ogni nodo ad ogni ciclo di simulazione
- ad ogni ciclo ogni nodo può eseguire più di un protocollo
- i protocolli vengono eseguiti sequenzialmente

L'INTERFACCIA CDPROTOCOL

```
/**
 * Defines cycle driven protocols, that is, protocols that have a periodic
 * activity in regular time intervals.
 */
public interface CDProtocol extends Protocol
{

    /**
     * A protocol which is defined by performing an algorithm in more or less
     * regular periodic intervals.
     * This method is called by the simulator engine once in each cycle with
     * the appropriate parameters.
     *
     * @param node
     *         the node on which this component is run
     * @param protocolID
     *         the id of this protocol in the protocol array
     */
    public void nextCycle(Node node, int protocolID);
}
```

Per definire un protocollo, l'utente deve

- implementare l'interfaccia **CDProtocol**, specificando il corpo del metodo `nextCycle()`
- il metodo viene chiamato dal simulatore ad ogni ciclo ed i parametri vengono passati direttamente dal simulatore

L'INTERFACCIA CDPROTOCOL

```
/**
 * Defines cycle driven protocols, that is, protocols that have a periodic
 * activity in regular time intervals.
 */
public interface CDProtocol extends Protocol
{
    /**
     * A protocol which is defined by performing an algorithm in more or less
     * regular periodic intervals.
     * This method is called by the simulator engine once in each cycle with
     * the appropriate parameters.
     *
     * @param node
     *         the node on which this component is run
     * @param protocolID
     *         the id of this protocol in the protocol array
     */
    public void nextCycle(Node node, int protocolID);
}
```

vengono passati dal simulatore:

- il nodo della rete su cui si esegue il protocollo
- l'identificatore attribuito al protocollo nel file di configurazione. Questo identificatore viene utilizzato per identificare i parametri del protocollo nel file di configurazione

L'INTERFACCIA LINKABLE

```
public interface Linkable extends Cleanable {
    /**
     * Returns the size of the neighbor list.
     */
    public int degree();

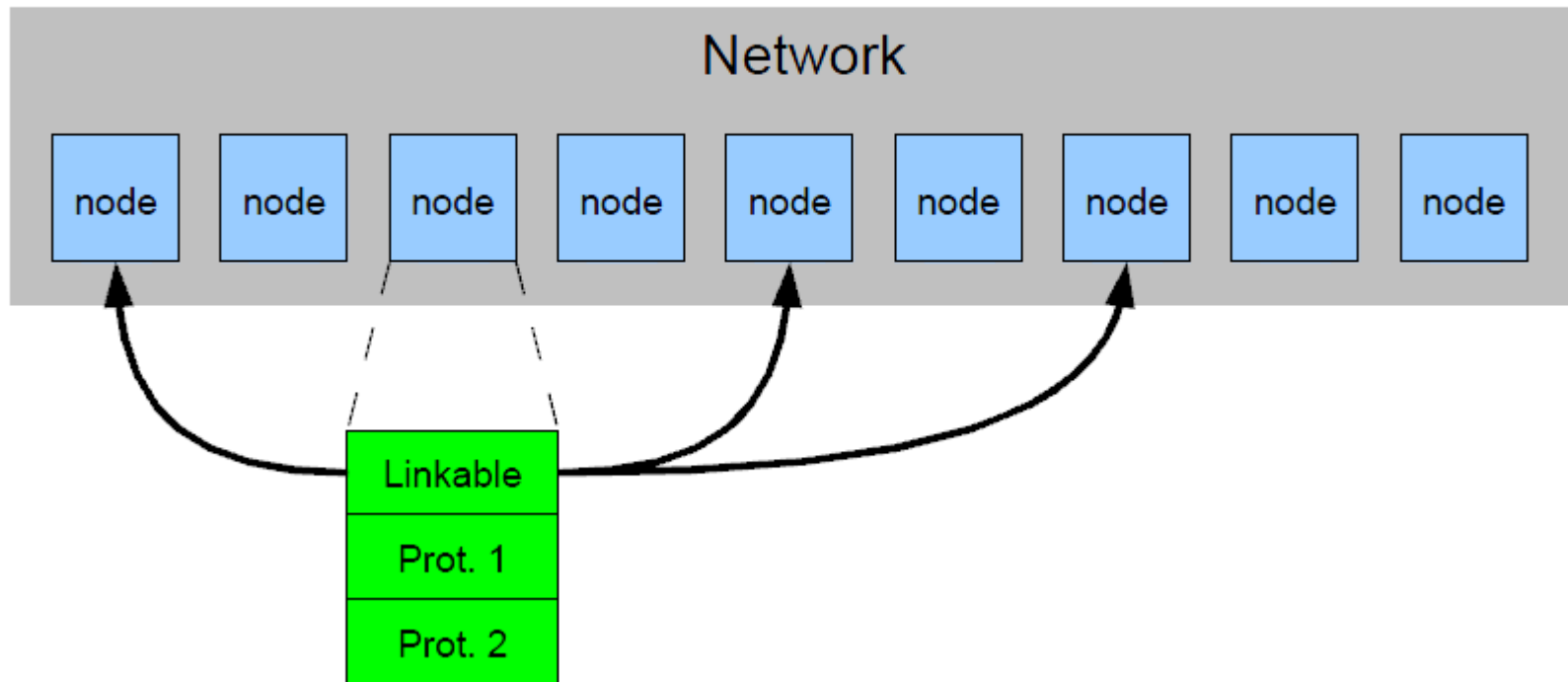
    /**
     * Returns the neighbor with the given index.
     */
    public Node getNeighbor(int i);

    /**
     * Add a neighbor to the current set of neighbors.
     */
    public boolean addNeighbor(Node neighbour);

    /**
     * Returns true if the given node is a member of the neighbor set.
     */
    public boolean contains(Node neighbor);

    /**
     * A possibility for optimization. An implementation should try to
     * compress its internal representation. Normally this is called
     * by initializers or other components when
     * no increase in the expected size of the neighborhood can be
     * expected.
     */
    public void pack();
}
```

L'INTERFACCIA LINKABLE



L'INTERFACCIA LINKABLE

Linkable utilizzata per gestire la 'vista' di un nodo

- Metodi definiti:
 - aggiunta di un vicino
 - ricerca di un vicino: restituisce un puntatore al vicino
 - grado di un nodo
- L'interfaccia non definisce metodi per la rimozione dei nodi: è necessario dare una propria definizione

L'INTERFACCIA CONTROL

- Consente di definire un insieme di **controlli**, ovvero operazioni che richiedono una **conoscenza globale della rete**
- Un controllo può essere di uno dei seguenti tipi
 - **Initializers** eseguiti **all'inizio della simulazione** per definire
 - la topologia iniziale dell'overlay
 - lo stato iniziale dei nodi
 - **Dynamics** eseguiti **periodicamente** durante la simulazione per
 - aggiungere nodi all'overlay
 - rimuovere nodi dall'overlay
 -
 - **Observers** eseguiti **periodicamente** durante la simulazione
 - Aggregazione dei valori dei diversi nodi (grado medio, coefficiente clustering,...)
 - ...

L'INTERFACCIA CONTROL

```
/**
 * Generic interface for classes that are responsible for observing or modifying
 * the ongoing simulation. It is designed to allow maximal flexibility therefore
 * poses virtually no restrictions on the implementation.
 */
public interface Control
{

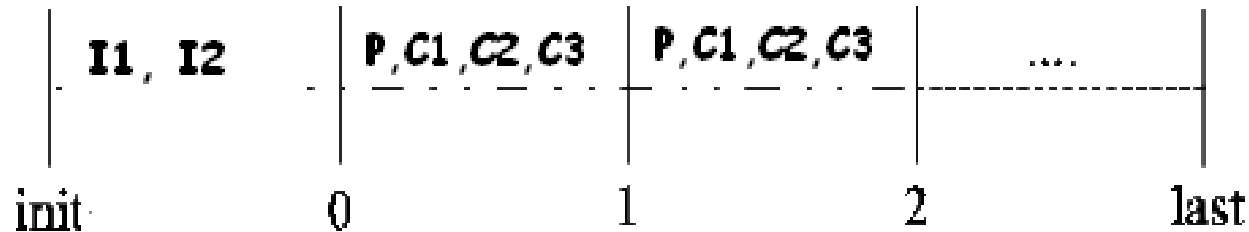
    /**
     * Performs arbitrary modifications or reports arbitrary information over the
     * components.
     * @return true if the simulation has to be stopped, false otherwise.
     */
    public boolean execute();

}
```

STRUTTURA GENERALE DEL SIMULATORE

```
for i := 1 to simulation.experiments do
  create Network
  create prototype Node :
    for i := 1 to #protocols do
      create protocol instance
  for j := 1 to network.size do
    clone prototype Node into Network
  create controls ( initializers, dynamics, observers )
  execute initializers
  for k := 1 to simulation.cycles do
    for j := 1 to network.size do
      for p := 1 to #protocols do
        execute Network.get(j).getProtocol(p).nextCycle()
      execute controls
    if ( one control returned true ) then
      break
```

PEERSIM: CYCLE-BASED SCHEDULING



Cycle-Based Simulation : supponiamo che sia stato definito

- due inizializzatori I1, I2
- un protocollo P
- tre diversi **controlli periodici** C1, C2, C3
- Dopo la fase di inizializzazione, il simulatore
 - **esegue n cicli di simulazione** (n specificato dall'utente)
 - ad ogni ciclo vengono eseguiti tutti i protocolli (nel nostro caso solo P) su tutti i nodi, e quindi tutti i controlli periodici (nel nostro caso C1, C2, C3)
 - è possibile configurare il simulatore in modo da stabilire l'ordine con cui vengono eseguiti i protocolli/controlli

PEERSIM: IL FILE DI CONFIGURAZIONE

Dopo che sono stati implementati tutti i protocolli e tutti i controlli, occorre 'comporre' le diverse entità per definire la simulazione. Occorre definire:

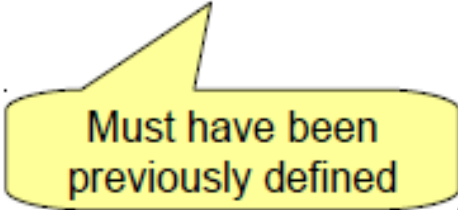
- quali componenti si utilizzano
- come interagiscono i diversi componenti

In Peersim una simulazione viene specificata mediante il file di configurazione un file di testo contenente

- **parametri globali**: dimensione della rete, il numero di cicli della simulazione,...
- **protocolli e controlli** che compongono la simulazione. Per ogni componente:
 - **riferimento alla classe** definita dall'utente o predefinita che implementa la componente
 - **parametri per la configurazione della componente**

PEERSIM: FILE DI CONFIGURAZIONE

- File di testo contenente un insieme di coppie (chiave, valore)
- Sintassi per la definizione di protocolli/controlli
`{protocol,init,control}.string_id [fullpath]classname`
string_id: identificatore del protocollo implementato dalla classe
[fullpath]classname
- Sintassi per la definizione dei parametri dei protocolli/controlli
`{protocol,init,control}.string_id.parameter_name`
parameter_value



Must have been
previously defined

FILE DI CONFIGURAZIONE: PARAMETRI GLOBALI

```
random.seed 1234567891
```

Global property: used to initialize the RNG

```
control.shf Shuffle
```

Shuffles the order in which nodes are visited at each cycle

```
simulation.cycles 100
```

Maximum number of simulation cycles

```
simulation.experiments 50
```

```
network.size 10^6
```

Number of nodes in the network

```
network.node peersim.core.GeneralNode
```

PEERSIM CYCLE DRIVEN: UN SECONDO ESEMPIO

Supponiamo di voler simulare un algoritmo distribuito di aggregazione di gossiping tra nodi

- Dimensione della rete: 50000 nodi
- Stato iniziale dei nodi: valore intero nell'intervallo (0-100)
- Protocollo:
 - Calcolo distribuito di una funzione (media) calcolata sull'insieme di valori memorizzati nei nodi della rete
 - Gossip Based Aggregation: ogni nodo seleziona periodicamente un vicino e scambia con esso l'approssimazione del valore calcolato. Entrambe i nodi aggiornano l'approssimazione corrente
- Topologia: Connessione casuale tra i nodi

PEERSIM: PROPRIETA' GLOBALI

```
01 # PEERSIM GOSSIP AGGREGATION
02
03 random.seed 1234567890
04 simulation cycles 30
05 control.shf Shuffle
06 network.size 50000
```

Proprietà globali

- **simulation cycles** cicli di simulazione
- **network size** dimensione della rete
- **shuffle** cambia l'ordine con cui i nodi vengono considerati ad ogni ciclo della simulazione
- **random seed** parametro utilizzato per **replicare esattamente** i risultati della simulazione basandosi su un comportamento pseudo-random

PEERSIM: I PROTOCOLLI

```
07  protocol.lnk IdleProtocol
08
09  protocol.avg example.aggregation.AverageFunction
10  protocol.avg.linkable lnk
```

Dichiarazione di Protocolli

`protocol.string_id [full_path]classname`

- assegna l'identificatore `string_id` al protocollo definito nella classe `classname`
- `protocol.lnk IdleProtocol` assegna il nome `lnk` al protocollo `IdleProtocol`
- `IdleProtocol`
 - protocollo `predefinito` da Peersim, implementa la interfaccia `Linkable`
 - è un contenitore di links
 - non è eseguibile

PEERSIM: I PROTOCOLLI

```
07 protocol.lnk IdleProtocol
08
09 protocol.avg example.aggregation.AverageFunction
10 protocol.avg.linkable lnk
```

Dichiarazione di protocolli:

protocol.string_id [full_path]classname

- assegna l'identificatore `string_id` al protocollo definito nella classe `classname`
- `example.aggregation.AverageFunction`
 - protocollo definito dall'utente
 - implementa un protocollo di aggregazione basato su gossiping
 - viene eseguito ad ogni ciclo
 - sfrutta il protocollo `lnk` per memorizzare la vista della rete

PEERSIM: I PROTOCOLLI

```
07 protocol.lnk IdleProtocol
08
09 protocol.avg example.aggregation.AverageFunction
10 protocol.avg.linkable lnk
```

Configurazione dei Protocolli:

- **protocol.string_id.xxx** definisce il parametro xxx del protocollo identificato da string
- **protocol.avg.linkable lnk** configura il protocollo avg in modo che utilizzi protocollo **lnk**, cioè l'**IdleProtocol**, per rappresentare la visione locale che ogni nodo possiede della rete
- il protocollo avg utilizza come overlay la topologia rappresentata in IdleProtocol

PEERSIM: INIZIALIZZATORI

```
11 init.rnd WireKOut
12 init.rnd.protocol lnk
13 init.rnd.k 20
```

Definizione delle componenti che devono essere eseguite **solo una volta per inizializzare la simulazione**

- **init.rnd WireKOut** associa al protocollo WireKOut (predefinito) l'identificatore rnd
 - **WireKOut** inizializza l'overlay mediante connessione casuale dei nodi ed utilizza come protocollo contenitore il protocollo lnk, cioè l'IdleProtocol.
- **init.rnd.k** Il parametro k definisce **il grado massimo** dei nodi dell'overlay. E' un parametro utilizzato da WireKOut
- Il protocollo rnd inizializza la vista e la memorizza nel 'protocollo contenitore' lnk

PEERSIM: ACQUISIZIONE DEI PARAMETRI

All'interno della classe WireKOut occorre acquisire i parametri

```
private final int k;

.....

private static final String PAR_DEGREE = "k";

.....

public WireKOut (String prefix)
{
    super(prefix);
    k = Configuration.getInt(prefix + "." + PAR_DEGREE);
}
```

- prefix : identificatore associato a WireKOut nel file di configurazione
 - viene passato automaticamente dal simulatore
- Le classi [Configuration](#) e [FastConfiguration](#) contengono i metodi per acquisire i parametri dal file di configurazione(vedere documentazione)

PEERSIM: I CONTROLLI

```
18 init.lin LinearDistribution
19 init.lin.protocol avg
20 init.lin.max 100
21 init.lin.min 1
```

Definizione delle componenti che devono essere eseguite **solo una volta per** inizializzare la simulazione

- **LinerDistribution** Libreria che definisce una distribuzione lineare crescente. I valori considerati sono compresi tra 1 e 100 (parametri **init.lin.min 1**, **init.lin.max 100**)
- **init.lin.protocol** avg definisce il protocollo che utilizza la distribuzione lineare dei valori

PEERSIM:I CONTROLLI

22 `control.avgo` `example.aggregation.AverageObserver`

23 `control.avgo.protocol` `avg`

- Definizione delle componenti che devono essere schedate periodicamente
- Le componenti seguono il monitoraggio dell'overlay
- `AverageObserver`: Componente predefinito che produce statistiche sui valori osservati sui nodi dell'overlay (vedere documentazione)
- Devo passare ad `AverageObserver` un 'puntatore' al protocollo `avg` (riferimento ad un oggetto di tipo `Protocol`) in modo che `AverageObserver` possa accedere alle strutture dati del protocollo e produrre statistiche

PEERSIM: FILE DI CONFIGURAZIONE

- E' possibile stabilire nel file di configurazione l'ordine con cui vengono eseguiti i protocolli
- se non esiste alcuna specifica, i protocolli vengono eseguiti **in ordine alfabetico**
- altrimenti:
 - con la clausola : **order.protocol P1 P2 P3**
i protocolli vengono eseguiti **in questo ordine P1 P2 e P3**. I protocolli non specificati vengono eseguiti in ordine alfabetico
 - con la clausola: **include.protocol P1 P2 P3**
eseguo P1 P2 e P3 in questo ordine, ma gli altri protocolli non vengono eseguiti
 - con la clausola: **control.shf Shuffle**
posso cambiare ad ogni ciclo di simulazione **l'ordine con cui il simulatore sceglie i nodi per l'esecuzione dei protocolli**
- i controlli vengono eseguiti comunque dopo i protocolli

PEERSIM: ESEGUIRE IL SIMULATORE

```
JAVA -cp <class-path> peersim.Simulator example1.txt
```

example1.txt è il nome del file di configurazione

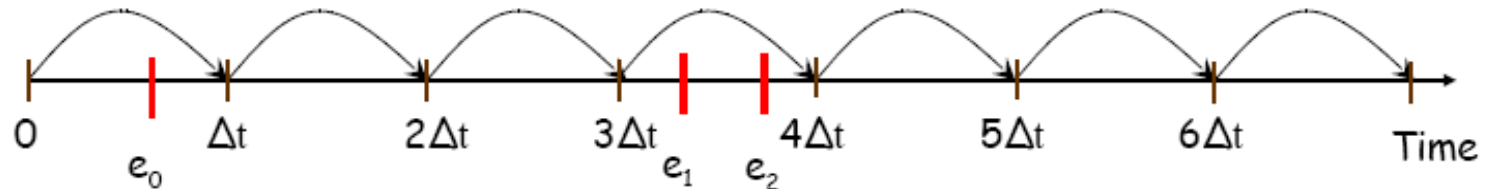
DISCRETE EVENT SIMULATION (DES)

- **Sistema:** insieme di entità interagenti che cooperano per fornire un insieme di funzionalità
 - esempio:
 - determinare quante classe veloci sono necessarie in un supermercato per assicurare un servizio 'veloce' a clienti con meno di 10 oggetti
 - entità: casse, clienti con meno di 10 oggetti
- **Stato del Sistema :**
 - insieme di variabili il cui valore caratterizza il sistema in un certo istante di tempo
 - esempio: numero di casse veloci, tempo di arrivo dei clienti con meno di 10 oggetti, numero di clienti in coda,...
- **Evento:** occorrenza istantanea di una qualsiasi azione che modifica lo stato del sistema
 - esempio: arrivo di un nuovo cliente, inizio del servizio da parte di una cassa, fine del servizio

DISCRETE EVENT SIMULATION

- **Simulation clock:** variabile che registra lo scorrere del tempo nella simulazione
 - E' necessario stabilire una relazione tra il tempo simulato ed il tempo reale (es: 1 tick= 200 ms)
 - in genere non esiste una relazione tra il tempo simulato ed il tempo necessario per l'esecuzione della simulazione
- Due approcci per l'avanzamento del tempo simulato:
 - ad **incrementi costanti**
 - Basato sull'occorrenza di eventi (**discret event simulation**)

AVANZAMENTO AD INCREMENTI COSTANTI

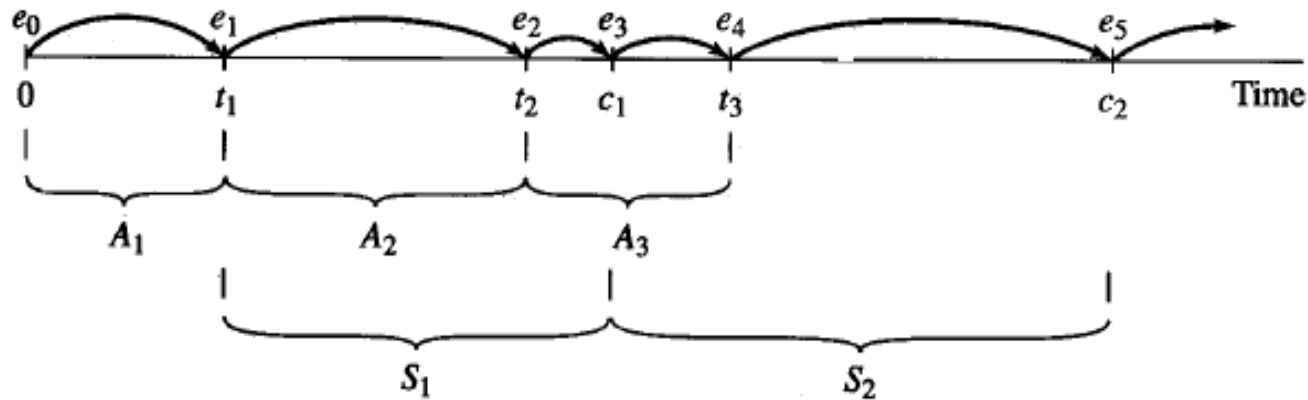


- L'avanzamento del tempo di simulazione avviene mediante **incrementi costanti (time step)**
- L'occorrenza di un qualsiasi evento viene fatta coincidere con l'inizio di un time step
- Gli eventi che avvengono durante un timestep vengono spostati e fatti coincidere con l'inizio del time step successivo
- Simulazione semplice da implementare, ma poco accurata

DISCRETE EVENT SIMULATION

- il tempo simulato viene inizializzato a 0
- si individuano gli eventi che possono modificare lo stato del sistema
- si associa ad ogni evento il suo **tempo di occorrenza** (timestamp)
- il simulatore mantiene una **lista degli eventi ordinata per valori crescenti di event time step**
- L'avanzamento del tempo viene guidato dai timestamps associati agli eventi
- Prossimo evento= Evento con timestamp minimo
- ad ogni passo di simulazione il tempo assume il valore del timestamp del **prossimo evento**
- il tempo 'salta' da un timestamp all'altro e 'non esiste' nell'intervallo tra due eventi successivi.
- i periodi in cui non avviene nessun evento rilevante per la simulazione vengono ignorati

DISCRETE EVENT SIMULATION



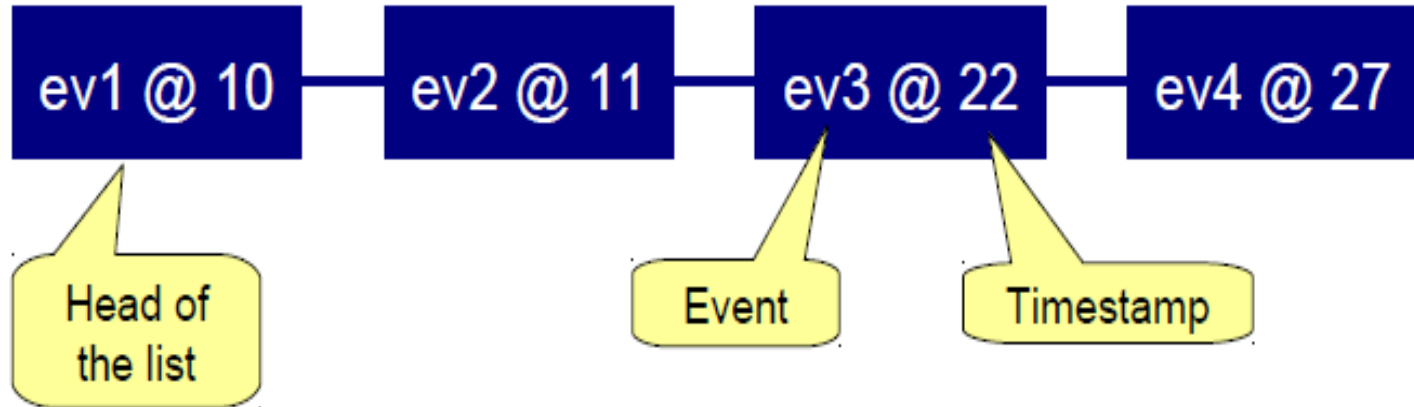
Esempio: Casse veloci e clienti

- t_i = tempo di arrivo dell' i -esimo cliente ($t_0=0$)
- $A_i = t_i - t_{i-1}$ = tempo di interarrivo tra due clienti
- S_i = tempo di servizio dell' i -esimo cliente
- c_i = istante di tempo in cui il cliente lascia il supermercato perchè ha ricevuto il servizio

EVENT DRIVEN PEERSIM

- Peersim definisce un insieme di interfacce/classi per il supporto della simulazione basata su eventi
- Peersim discretizza il tempo in istanti virtuali o **tick**
- In base ai tick si misura
 - la durata dell'intera simulazione
 - il tempo a cui deve avvenire qualsiasi evento
- Il simulatore consente di generare eventi e di indicare il ciclo a cui un evento deve essere eseguito
- Tipi di eventi in Peersim
 - **eventi periodici**
 - spedizione del proprio stato ad un vicino
 - **eventi asincroni**
 - ricezione di un messaggio da un vicino

EVENT DRIVEN PEERSIM



Nel paradigma event-driven il simulatore gestisce una coda di eventi, contenente gli eventi (invio di un messaggio,...) con associato il timestamp

EVENTI PERIODICI: SCHEDULAZIONE

eventi periodici

- un protocollo può essere schedulato periodicamente, anche se si usa la modalità event based (come nella simulazione cycle based)
- poichè **non esiste il concetto di ciclo**, occorre:
 - specificare l'istante temporale (tick) in cui l'evento viene schedulato per la prima volta
 - l'intervallo di tempo (STEP) che intercorre tra due successive generazioni dell'evento.
 - come nella simulazione cycle-based, il codice del protocollo va comunque specificato mediante l'implementazione del metodo **nextcycle()** dell'interfaccia CDProtocol

EVENTI PERIODICI: SCHEDULAZIONE

- Si può utilizzare uno **schedulatore predefinito**, il CDScheduler, per inserire nella coda degli eventi la prima esecuzione di un protocollo periodico
 - CDScheduler implementato in una classe predefinita di Peersim
- Al termine della esecuzione, il protocollo stesso si schedula per l'esecuzione successiva
 - il tick a cui avviene l'esecuzione successiva viene stabilito mediante **lo step** specificato nel file di configurazione
- Quando viene schedulato l'evento corrispondente al protocollo periodico, viene eseguito il corpo della **nextcycle ()** di quel protocollo
- **nextcycle(Node, ProtocolID)**, parametri passati dal simulatore
 - il nodo della rete su cui si esegue il protocollo
 - l'identificatore attribuito al protocollo nel file di configurazione. Questo identificatore viene utilizzato per identificare i parametri del protocollo nel file di configurazione

EVENTI PERIODICI: FILE DI CONFIGURAZIONE

```
01 N 4560
02 M 20
03 simulation.endtime N
04 protocol.c1 AverageProtocol
05 protocol.c1.step M
06 init.schedulatore CDScheduler
07 init.schedulatore.protocol c1
```

- **clausola 06:** Si attribuisce il nome schedulatore allo schedulatore standard di Peersim, contenuto nella classe CDScheduler
- **clausola 07:** Si indica che lo schedulatore deve schedulare la prima esecuzione del protocollo il cui nome simbolico attribuito nel file di configurazione è c1
- **clausola 05:** si indica che quel protocollo deve essere schedulato ogni 20 tick() del tempo virtuale

EVENT BASED PEERSIM

Per generare un nuovo evento occorre:

- definire l'evento specificando
 - il **delay** dell'evento, cioè l'intervallo di tempo (rispetto al tempo attuale) che deve trascorrere prima che l'evento venga schedulato
 - l'**oggetto** che **descrive** l'evento
 - il **nodo n** a cui l'evento è destinato
 - il **protocollo** in esecuzione su n che riceverà l'evento
- inserire l'evento nella coda degli eventi gestita dal simulatore
 - metodo **add** della classe **EDSimulator**
- definire l'**handler** dell'evento

GENERAZIONE DEGLI EVENTI

Per la generazione degli **eventi periodici**, occorre

- implementare l'interfaccia **CDProtocol**, definendo il corpo del metodo **nextCycle**, come nell'approccio cycle based
- fornire uno schedulatore di eventi periodici che
 - associ un time stamp alla prima occorrenza dell'evento
 - definisca la frequenza di schedulazione dell'evento
- Peersim fornisce uno schedulatore predefinito (**CDScheduler**)
- è possibile implementare schedulatori ad hoc

GESTIONE DEGLI EVENTI

Per la gestione degli eventi occorre implementare opportuni event-handlers

Per implementare un **event handler** occorre implementare l'interfaccia **EDProtocol**

- definire il corpo del metodo **processEvent**.
- il metodo **processEvent()** viene invocato su un nodo, per un certo protocollo e contiene il codice da eseguire al momento della **ricezione di** evento (handler dell'evento)

EVENT DRIVEN PEERSIM

```
for i := 1 to simulation.experiments do
  initialize MinHeap events
  create Network
  create prototype Node :
    for i := 1 to #protocols do
      create protocol instance
  for j := 1 to network.size do
    clone prototype Node into Network
  createcontrols ( initializers , dynamics , observers )
  execute      initializers
  time = 0
  while ( time < simulation.endtime) do
    (node, pid, e) = events.getMin( ) ;
    node.getProtocol(pid).processEvent( node, pid, event)
    if (event is acontrol that returned true) then
      break
```

INVIO/RICEZIONE DI MESSAGGI

- L'invio/ricezione dei messaggi è modellato mediante eventi
- Ogni nodo può
 - spedire messaggi ad un altro nodo.
 - un messaggio è spedito ad un protocollo in esecuzione su un nodo
 - ad ogni messaggio viene associato un timestamp che indica l'istante temporale in cui tale messaggio dovrà essere ricevuto
 - il timestamp può essere utilizzato per modellare le latenze della rete
 - ricevere messaggi.
- I protocolli che ricevono messaggi devono implementare un handler che deve essere eseguito al momento della ricezione del messaggio
 - Handler= implementazione del metodo `processEvent()` della classe `EDSimulator`

IL LIVELLO DI TRASPORTO

- La spedizione del messaggio viene gestita da **protocolli** associati ai nodi che simulano il **livello di trasporto della rete**
- Peersim offre diversi livelli di trasporto implementati da protocolli predefiniti
 - **UniformRandomTransport**. Servizio di trasporto affidabile, associa **latenze variabili** alla trasmissione dei messaggi
 - **UnreliableTransport**. Latenze variabili + Perdita di messaggi con una data probabilità
 - altri protocolli.....
 - inoltre è possibile definire nuovi servizi di trasporto implementando l'interfaccia **Transport**

IL LIVELLO DI TRASPORTO

Un protocollo che esegue le send e della receive utilizza un certo livello di trasporto

`protocol.urt` UniformRandomTransport

`protocol.urt.mindelay` MIND

`protocol.urt.maxdelay` MAXD

`protocol.transp` UnreliableTransport

`protocol.transp.transport` urt

`protocol.transp.drop` DROP

`protocol.example` Example

`protocol.example` transp

IL LIVELLO DI TRASPORTO

Un protocollo che esegue le send e della receive utilizza un certo livello di trasporto

`protocol.urt` UniformRandomTransport

`protocol.urt.mindelay` MIND

`protocol.urt.maxdelay` MAXD

con queste clausole si definisce

- il protocollo `urt` che è implementato mediante la classe predefinita da Peersim `UniformRandomTransport`
- `UniformRandomTransport` introduce latenze variabili nell'invio dei messaggi
 - i parametri `MIND` a `MAXD` indicano la latenza minima e massima

IL LIVELLO DI TRASPORTO

Un protocollo che esegue le send e della receive utilizza un certo livello di trasporto

`protocol.transp` UnreliableTransport

`protocol.transp.transport` urt

`protocol.transp.drop` DROP

con queste clausole si definisce

- Il protocollo `UnreliableTransport` che implementa un livello di trasporto non affidabile
- `UnreliableTransport` introduce perdita di messaggi
- Occorre specificare i paramentri del protocollo transp
 - `urt` specifica che il protocollo UnrelaibleTrasport è costruito sul protocollo
 - `drop` indica la probabilità di perdita di messaggi

IL LIVELLO DI TRASPORTO

Il protocollo Example utilizza il livello di Trasporto

```
protocol.example Example
```

```
protocol.example.transport transp
```

- con la seconda clausola si specifica che il protocollo example, implementato nella classe Example, utilizza il protocollo transp, come protocollo per il livello di trasporto
- Il protocollo specificato (transp) verrà utilizzato ogni volta che si effettua una send() (modella latenza e perdita di messaggi)

EVENT DRIVEN PEERSIM

Spedizione di messaggi

`Send(M, D, Mes, Prot)`

- M: Nodo Mittente
- D: Nodo Destinatario
- Mes: Messaggio
- Prot: Il messaggio viene consegnato al protocollo P in esecuzione sul nodo D

Ricezione di messaggi

- basata sulla definizione di `message handler`
- il protocollo che intende ricevere il messaggio deve definire un handler che si incarica di ricevere il messaggio e di elaborarlo
- simile agli event handler di JAVA (gestione di eventi generati dall'interfaccia,...)

GOSSIP BASED AVERAGE

Calcolo della media basato su tecniche di gossiping

- un valore intero V per ogni nodo della rete
- V inizializzato in **modo casuale**
- algoritmo eseguito da ogni nodo
 - scelta casuale di un vicino NB e scambio dei valori
 - aggiornamento del valore V con la media tra V ed il valore ricevuto dal vicino

La simulazione in Peersim richiede:

- la definizione di un protocollo **schedulato periodicamente** che contenga un metodo che implementi la scelta casuale di NB e l'invio di V ad NB
- la definizione di un handler che implementi:
 - la ricezione dei messaggi dai nodi vicini
 - l'eventuale invio del proprio valore al nodo vicino
 - aggiornamento della media.

EVENT DRIVEN SIMULATION

- Definizione del **messaggio** scambiato tra i nodi

```
class AverageMessage {  
    final double value;  
    public AverageMessage (double value, Node sender)  
    {  
        this.value = value;  
        this.sender = sender  
    }  
}
```

- Definizione di una classe che implementi sia CDPprotocol che EDProtocol
 - Schedulazione periodica della send
 - Definizione della **nextcycle()**
 - Definizione dell'handler associato alla ricezione dell'evento
 - Definizione della **processevent()**

GOSSIP BASED AVERAGE: EVENTI PERIODICI

```
public Class AverageED implements CDProtocol, EDProtocol
```

```
public AverageED {....}
```

```
public void nextCycle (Node node, int pid)
{ Linkable linkable = (Linkable) node.getProtocol
    (FastConfig.getLinkable(pid));
  if (linkable.degree( ) > 0
    { Node peern = <scelta casuale di un nodo vicino>
      Transport t = (Transport)
        node.getProtocol(FastConfig.getTransport(pid))
      t.send(node, peern, new AverageMessage(value, node),pid)
    }
}
```

GOSSIP BASED AVERAGE: MESSAGE HANDLER

```
public void processEvent (Node node, int pid, Object event)

{
    AverageMessage aem = (AverageMessage) event;
    // Questo metodo viene schedulato quando viene ricevuto un
    // messaggio
    // sul nodo Node, diretto al protocollo pid
    // l'istruzione precedente equivale ad una receive

    if (aem.sender != null)
    Transport t = ((Transport)
        node.getProtocol(FastConfig.getTransport(pid)))
    t.send (node, aem.sender, newAverageMeassage(value,null),pid);
    value = (value + aem.value) / 2
}
```

GOSSIP BASED AVERAGE: MESSAGE HANDLER

- il metodo `processEvent` gestisce i messaggi ricevuti da un protocollo associato ad un nodo
- `Mittente = null` utilizzato per implementare lo scambio di valori
 - se `mittante = null`, non si invia una risposta, perchè il messaggio ricevuto è già un messaggio di risposta
 - la risposta viene inviata accedendo al servizio di trasporto
- ad ogni protocollo che utilizza l'invio di messaggi deve essere associato un protocollo di trasporto
- `FastConfig.getTransport (pid)` restituisce un riferimento al protocollo di trasporto associato al protocollo `pid`
- L'associazione viene specificata nel file di configurazione

GOSSIP BASED AVERAGE: IL FILE DI CONFIGURAZIONE

Il file di configurazione deve contenere

- parametri globali: dimensione della rete, fine della simulazione,....
- definizione del protocollo che definisce l'overlay
- definizione dei protocolli di trasporto associati ai protocolli che utilizzano scambio di messaggi
- politica utilizzata per la schedulazione degli eventi periodici
-

CONFIGURAZIONE DELLA SIMULAZIONE

```
01 # PEERSIM EVENT DRIVEN AGGREGATION
02 SIZE 10^3
03 network.size SIZE
04 random.seed 1234567890
05 simulation.endtime 10^6
06 network.node peersim.core.GeneralNode
```

Proprietà globali

- **network.size** dimensione della rete
- **network.node** classe che implementa l'oggetto nodo

La simulazione termina quando

- la coda degli eventi è vuota
- tutti gli eventi nella coda sono caratterizzati da un
time-stamp > simulation.endtime

CONFIGURAZIONE DELLA SIMULAZIONE

```
07  #protocols
08  CYCLE 1000
09  protocol.avg AverageED
10  protocol.avg.linkable link
11  protocol.avg.step CYCLE
12  protocol.avg.transport tr
```

Proprietà dei protocolli

- al protocollo implementato nella classe *AverageED* viene assegnato il nome *avg*
- il protocollo utilizzato da *avg* per la gestione dell'overlay è *link*
- il protocollo utilizzato da *avg* per l'invio dei messaggi è *tr*
- il protocollo *avg* deve essere eseguito con frequenza *CYCLE*
- il protocollo *tr* deve essere opportunamente configurato (vedi pagina successiva)

CONFIGURAZIONE DELLA SIMULAZIONE

```
13 MINDELAY 10
14 MAXDELAY 400
15 DROP 20
16 protocol.urc UniformRandomTransport
17 protocol.urc.mindelay (CYCLE*MINDELAY) / 100
18 protocol.urc.maxdelay (CYCLE*MAXDELAY) / 100
19 protocol.tr UnreliableTransport
20 protocol.tr.transport urc
21 protocol.tr drop DROP
```

Configurazione dei protocolli di trasporto

- si definisce il protocollo urc che assegna ai messaggi un **valore casuale** della latenza variabile tra mindelay e maxdelay
- si definisce un **protocollo wrapper** di urc, il protocollo tr, che aggiunge alle funzionalità di urc lo scarto di messaggi con probabilità DROP

CONFIGURAZIONE DELLA SIMULAZIONE

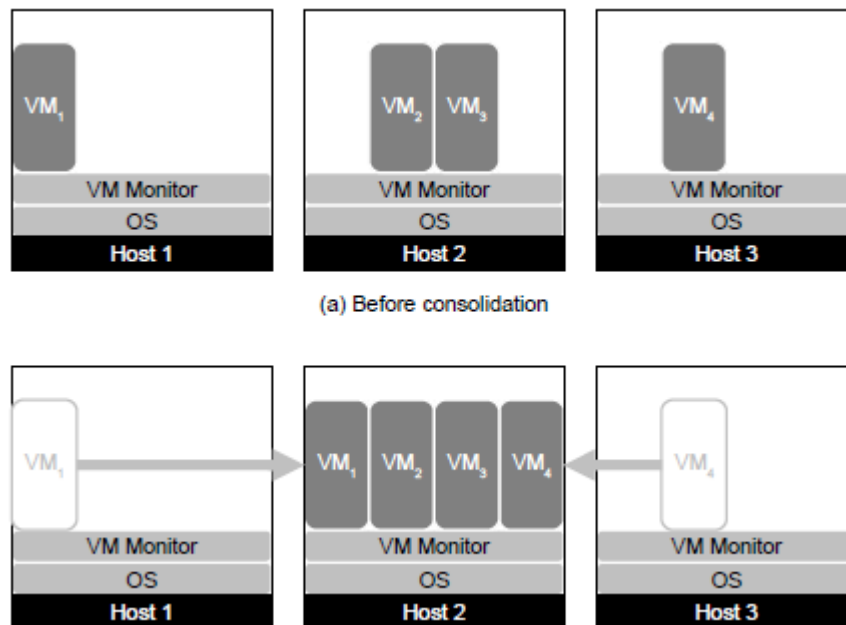
```
21 init.sch CDScheduler
22 init.sch.protocol avg
23 init.sch.randstart
```

Definizione di uno schedulatore per gli eventi periodici

- Il protocollo avg implementa il metodo `nextCycle` e contiene quindi un evento che deve essere schedulato periodicamente
- Occorre definire un componente che effettui lo scheduling periodico del protocollo
- Il componente scelto è uno schedulatore predefinito di Peersim: **CDScheduler**
- CDScheduler viene associato al protocollo avg
- La prima invocazione del protocollo `nextCycle` avviene in un istante di tempo casuale tra 0 e **CYCLE**
- Successivamente il metodo viene invocato ogni **CYCLE** time steps

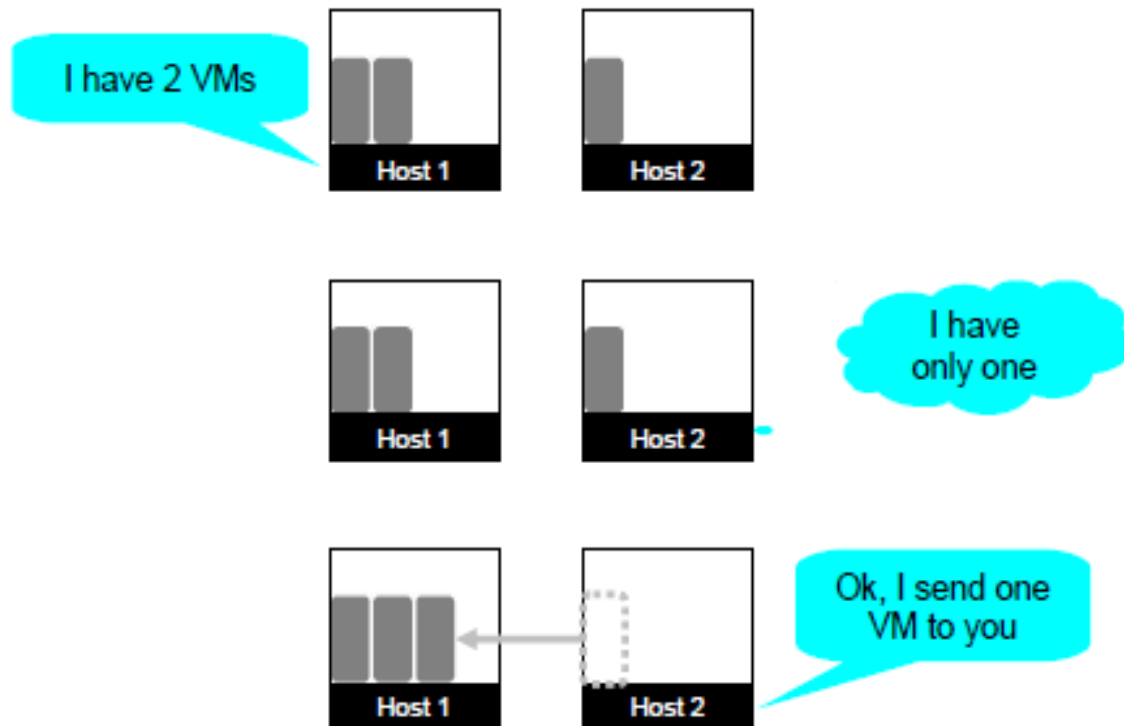
UN ESEMPIO DI SIMULAZIONE: VMAN

- Scopo del protocollo VMAN: ridurre il consumo di energia in architetture cloud
- idea:
 - migrazione di macchine virtuali da server poco carichi verso server più carichi
 - i server a cui non è associata alcuna VM possono essere messi in stand-by



UN ESEMPIO DI SIMULAZIONE: VMAN

Implementazione di VMAN mediante *gossip*



UN ESEMPIO DI SIMULAZIONE: VMAN

- Ogni nodo n_i memorizza nella variabile H_i il numero di virtual machines che sono in esecuzione su n_i
 - Tutti i nodi hanno capacità massima C
- Il nodo n_i seleziona in maniera random un nodo n_j
 - se $H_i \leq H_j$ allora il nodo n_i invia una VM al nodo n_j
 - se $H_i > H_j$ allora riceve una VM dal nodo n_j

IL FILE DI CONFIGURAZIONE

```
simulation.cycles 20
simulation.experiments 10
network.size 10000
WIREK 20
CORES 8
```

General settings

```
random.seed 1234567890
```

```
protocol.lnk example.newscast.SimpleNewscast
protocol.lnk.cache WIREK
protocol.vman example.vman.BasicConsolidation
protocol.vman.capacity CORES
protocol.vman.linkable lnk
```

Protocols settings

```
init.rnd WireKOut
init.rnd.protocol lnk
init.rnd.k WIREK
```

Controls settings

```
init.ld example.vman.RandomDistributionInitializer
init.ld.protocol vman
init.ld.min 0
init.ld.max CORES
```

```
include.init rnd ld
```

```
control.shf Shuffle
control.vmo example.vman.VMObserver
control.vmo.protocol vman
```


PEER: INIZIALIZZAZIONE DELLO STATO

```
public class RandomDistributionInitializer implements Control, NodeInitializer {  
  
    // ... constants and local variables omitted ...  
  
    /**  
     * This class provides a simple random distribution in a bounded  
     * interval defined by parameters {@link #PAR_MIN} and {@link #PAR_MAX}.  
     */  
    public boolean execute() {  
        int tmp;  
        for (int i = 0; i < Network.size(); ++i) {  
            initialize( Network.get(i) );  
        }  
        return false;  
    }  
  
    /**  
     * Initialize a single node by allocating a random number of virtual  
     * machines, defined by parameters {@link #PAR_MIN} and {@link #PAR_MAX}.  
     */  
    public void initialize(Node n) {  
        int tmp = min + CommonState.r.nextInt( max-min+1 );  
        assert( tmp >= min );  
        assert( tmp <= max );  
        ((SingleValue) n.getProtocol( protocolID )).setValue( tmp );  
    }  
}
```

All'inizio ad ogni nodo vengono attribuite un numero casuale di VM

BASICCONSOLIDATION: IL PROTOCOLLO DI GOSSIP

```
public class BasicConsolidation extends SingleValueHolder implements CDPProtocol
{
    /**
     * Node capacity. The capacity is the maximum number of
     * Virtual Machines that a node can host. Defaults to 1.
     *
     * @config
     */
    protected static final String PAR_CAPACITY = "capacity";

    /** Capacity. Obtained from config property {@link #PAR_CAPACITY}. */
    private final int capacity_value;

    /**
     * Standard constructor that reads the configuration parameters. Invoked by
     * the simulation engine.
     *
     * @param prefix
     *         the configuration prefix for this class.
     */
    public BasicConsolidation(String prefix) {
        super(prefix);
        // get capacity value from the config file. Default 1.
        capacity_value = (Configuration.getInt(prefix+"."+PAR_CAPACITY, 1));
    }
}
```

BASICCONSOLIDATION: IL PROTOCOLLO DI GOSSIP

```
/**
 * Using an underlying {@link Linkable} protocol
 * performs a consolidation step with all neighbors of the node
 * passed as parameter.
 *
 * @param node
 *         the node on which this component is run.
 * @param protocolID
 *         the id of this protocol in the protocol array.
 */
public void nextCycle(Node node, int protocolID) {
    int linkableID = FastConfig.getLinkable(protocolID);
    Linkable linkable = (Linkable) node.getProtocol(linkableID);

    for (int i = 0; i < linkable.degree(); ++i) {
        Node peer = linkable.getNeighbor(i);
        // The selected peer could be inactive
        if (!peer.isUp())
            continue;
        BasicConsolidation n = (BasicConsolidation) peer.getProtocol(protocolID);
        doTransfer(n);
    }
}
```

BASICCONSOLIDATION: IL PROTOCOLLO DI GOSSIP

```
/**
 * Performs the actual consolidation selecting to make a PUSH or PULL
 * approach. The idea is to send the maximum number of VMs from
 * the node with fewer VMs to the other one.
 *
 * @param neighbor
 *         the selected node to talk with.
 */
protected void doTransfer(BasicConsolidation neighbor) {
    int a1 = (int)this.value;
    int a2 = (int)neighbor.value;
    if ( a1 == 0 || a2 == 0 ) return;
    int a1_avail = capacity_value - a1;
    int a2_avail = neighbor.capacity_value - a2;
    int trans = Math.min( Math.min(a1,a2),
                          Math.min(a1_avail, a2_avail) );

    if (a1 <= a2) {
        // PUSH
        a1 -= trans;
        a2 += trans;
    } else {
        // PULL
        a1 += trans;
        a2 -= trans;
    }
    assert( a1 >= 0 && a1 <= capacity_value );
    assert( a2 >= 0 && a2 <= capacity_value );
    this.value = (float)a1;
    neighbor.value = (float)a2;
}
```

VIRTUAL MACHINE OBSERVER

- Si implementa un controllo che permette di stampare ad ogni ciclo il numero di server che possiedono esattamente k VM
- Il controllo
 - implementa l'interfaccia Control
 - eseguito periodicamente

VIRTUAL MACHINE OBSERVER

```
public class VMObserver implements Control {

    private static final String PAR_PROT = "protocol";
    private final String name;
    private final int pid;

    public VMObserver(String name) {
        this.name = name;
        pid = Configuration.getPid(name + "." + PAR_PROT);
    }

    public boolean execute() {
        IncrementalFreq freqs = new IncrementalFreq();
        long time = peersim.core.CommonState.getTime();
        int capacity = 0;
        for (int i = 0; i < Network.size(); i++) {
            BasicConsolidation protocol = (BasicConsolidation)
Network.get(i).getProtocol(pid);
            capacity = protocol.getCapacity();
            freqs.add((int)protocol.getValue());
        }
        System.out.print(name + ": " + time);
        for (int j=0; j<=capacity; ++j )
            System.out.print(" " + freqs.getFreq(j));
        System.out.println();
        return false;
    }
}
```

PEERSIM: DOCUMENTAZIONE

- PeerSim Web Page

<http://peersim.sf.net/>

- Class documentation

<http://peersim.sourceforge.net/doc/index.html>

- Tutorial for the Cycle-based engine

<http://peersim.sourceforge.net/tutorial1/tutorial1.pdf>